

## Criteria A1 – Analyzing the Problem

### Describing the Problem

Our school has lots of computers, as well as a SmartBoard in every classroom. The teachers keep trying to find clever ways to use the computers to improve classroom instruction. They look for “educational software”, like games and quizzes and videos and web-sites.

Ms Fizz is a math teacher at our school. She asked our Computer Science teacher whether the IB Computer Science students could create software for her to use on her SmartBoard. So I went to talk to Ms Fizz about some ideas. She said she wasn't interested in videos or that sort of thing, and that most web-sites were pretty useless. But her students need lots of drill and practice, and maybe she would be interested in programs that let the students do more drill and practice at home. She had seen lots of educational software, but mostly it was either for social studies – like multiple choice quizzes – or it was for **doing** math, like drawing graphs. She wanted a math **quiz** program that the students could use on their own, or maybe she could run it on the SmartBoard during class.

After a bit of discussion, and some web-surfing, Ms Fizz mentioned an old TV show called “Hollywood Squares” that was sort of a combination of Tic-Tac-Toe and Trivial Pursuit. She described it like this:

There is a Tic-Tac-Toe board. When the X-player chooses a square, they have to answer a question correctly in order to get their X on that square. Then it's O's turn. If the player answers incorrectly, they don't get the square. So you could end up with a board looking like this if O missed a question and X answered correctly 3 times:

	O	
X	X	X

Ms Fizz had the idea that the questions could be math problems. Then 2 students could play against each other, or she could split the class into teams that play against each other.

### Collecting Information

Since the game is for a math class, the questions need to be math problems (or questions about math vocabulary). I looked at some multiple-choice quizzes, some written tests, and some of the homework problems in a math text-book. Here are some sample questions:

<b>Quiz</b>	<b>Text</b>	<b>Homework</b>
(1) If the discriminant of a parabola is -4, how many roots does it have? (a) 0 (b) 1 (c) 2 (d) 3  (2) What is the sum of the roots of $x^2 - 4x + 2 = 0$ ? (a) 2 (b) 4 (c) 0 (d) 1  (3) Where is the vertex of the parabola $y = x^2 - 4x + 2$ ?	(1) Draw the graph of $y = \frac{x^2 - 4x + 3}{x - 3}$  (2) Find the intersection of $y = 2x + 3$ and $y = x^2 - 4$ . Show your work.  (3) Explain how the <b>discriminant</b> is used.	(1) Graph each parabola: (a) $y = x^2 - 4x + 3$ (b) $y = x^2 - 4x + 4$ (c) $y = x^2 - 4x + 5$  (2) For each graph in #1, state the number of roots.  (3) Write the formula of a parabola through (-1,4) , (0,1) and (1,0).

Ms Fizz was especially interested that the game be “fast” and “easy” and “fun”. She felt that her students would be more likely to use the game if the questions were quick and easy to answer, if typing the answers was easy, and if the game ended quickly so the students could play again.

It was apparent that not all the sample questions would be suitable for use in a computerized quiz game. For example, asking the players to draw a graph is probably out of the question.

- The questions should be short and have quick answers
- The students cannot be required to draw graphs or other pictures
- The answers should not require the students to write complex formulas
- Some special symbols, like powers, should be used in the problems, as computer symbols like  $x^2$  are not understandable to many students
- “Show your work” is not sensible

Looking at the sample questions, it seems the quiz questions were most suitable. Multiple-choice is particularly good, but simple numerical answers would also be acceptable.

### Systematic Analysis of Input, Output and Processing – Possible Difficulties

The appearance of the Tic-Tac-Toe board is pretty obvious, but the input and output of math formulas and symbols might cause some difficulties. And creating problems might be tricky.

Outputting Questions	Inputting Answers	Processing
Math formulas often contain strange symbols that cannot be typed on a keyboard. These can be created using ASCII or Unicode, but probably won't display “nicely”. Fractions present a large problem – might require graphics mode for printing. Surds are also difficult to print.	This has the same problems as outputting the questions, but is even more difficult because the users want to type their answers on the keyboard. They won't want to look up ASCII codes for special symbols. This might limit the types of question and answers that are possible. And typing cannot be done in graphics mode, at least not easily.	The questions need to come from somewhere. They cannot be hard-coded inside the program. They can either be stored in data-files or created by clever methods. For example, to produce a random parabola problem, the program can choose random numbers for the coefficients A, B, and C. But Ms Fizz says A, B, and C need to “fit together” - they can't just be chosen at random.
Simple exponents like squared and cubed can be printed using simple characters, but other exponents need to be “raised” above the x, perhaps requiring graphics mode.	What if the correct answer is a surd or a fraction like $1/3$ ? A decimal approximation won't be good enough.	Comparing users answers to correct answers can be difficult, as the user might type 0.3333 instead of $1/3$ .

Ms Fizz agreed to produce a list of sample problems to discuss. I warned her that some problems might not be suitable, as the printing and typing might be too difficult.

After some discussion, we agreed that multiple-choice questions were a good strategy, because they are quicker to answer and require very little typing. But I couldn't imagine how to program the computer to randomly generate 3 incorrect answers along with the right answer. So we agreed that the teacher would need to write the questions ahead of time rather than the computer “generating” the problems. True/false questions are possible, as well as simple numbers like whole numbers.

### Possible Advantages of a Computer System

Since some math formulas and problems might be excluded from the game, it's important to consider the advantages of using a computerized game.

- Students can practice as much as they wish, without needing the teacher around
- The computer gives immediate feedback about whether answers are correct
- Working with another student and playing a game should be fun and motivating, and encourage students to spend more time practicing than they would otherwise
- The teacher can use the game on the SmartBoard during class as a motivational tool

### Criteria A2 – Criteria for Success (preliminary)

Considering the analysis, the following goals seem sensible. The reasons in the chart were mentioned in the analysis above.

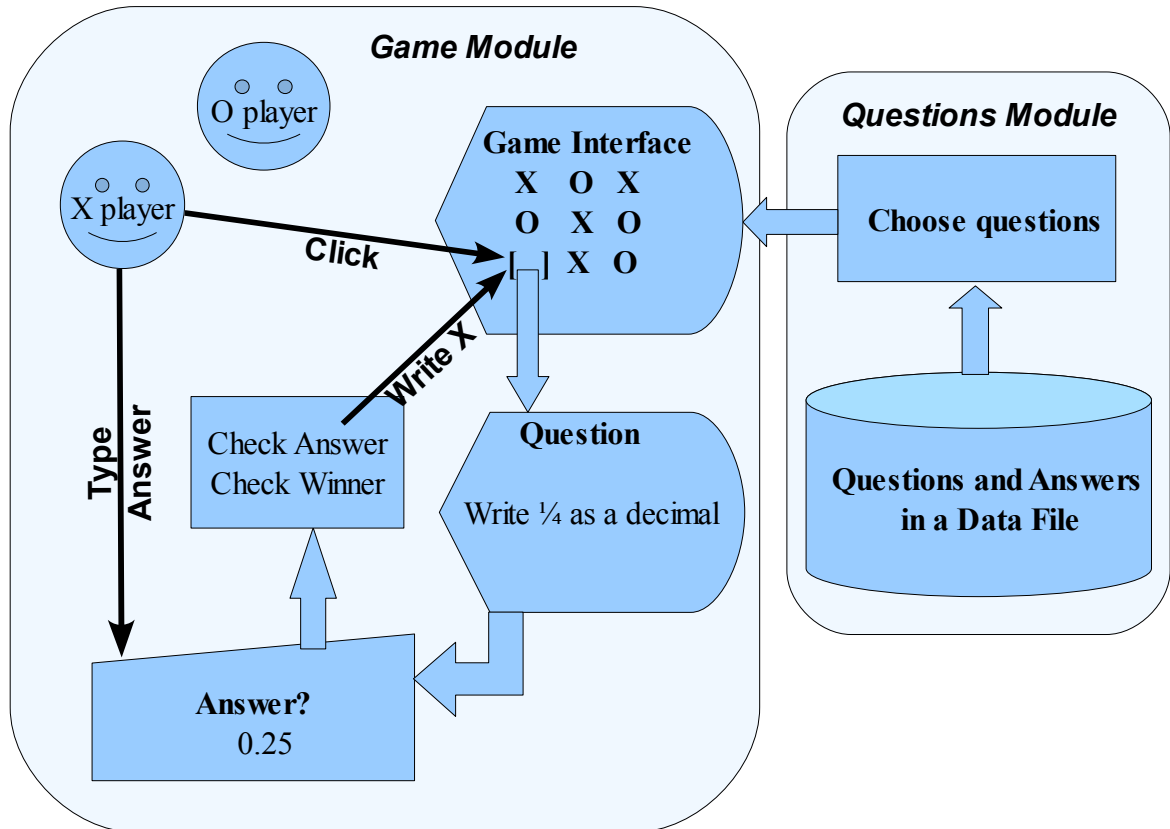
<b>Goal</b>	<b>Reason(s)</b>	<b>Limitation(s)</b>
Game plays Tic-Tac-Toe , using correct Tic-Tac-Toe rules	It's fun	3x3 board only
Each Tic-Tac-Toe square asks a question	That's the rules of the game	Questions will be selected from a list – not auto-generated
Questions should contain appropriate math content	Students should be learning and practicing their math	Some text-book problems are not suitable – e.g. complex formulas or graphs
Game should be quick, easy and satisfying, including a simple and clear user-interface	Motivate students to practice , more fun = more practice	Don't require complex input (fractions, complex formulas)
Teacher can create and save problems	Auto-generating problems is too difficult to program	No complex formulas Pictures?
Some special symbols can be used in the questions – squares, cubes, simple square-roots	Math without special symbols is difficult to read and understand	Many special symbols will not be implemented, especially fractions and complex surds like the discriminant in the quadratic formula

These were the preliminary goals, used to create the initial design and prototype. After discussing the prototype with the user, the goals were revised. The complete goals are presented below, following the prototype.

## Criteria A3 – Prototype Solution

### Initial Design

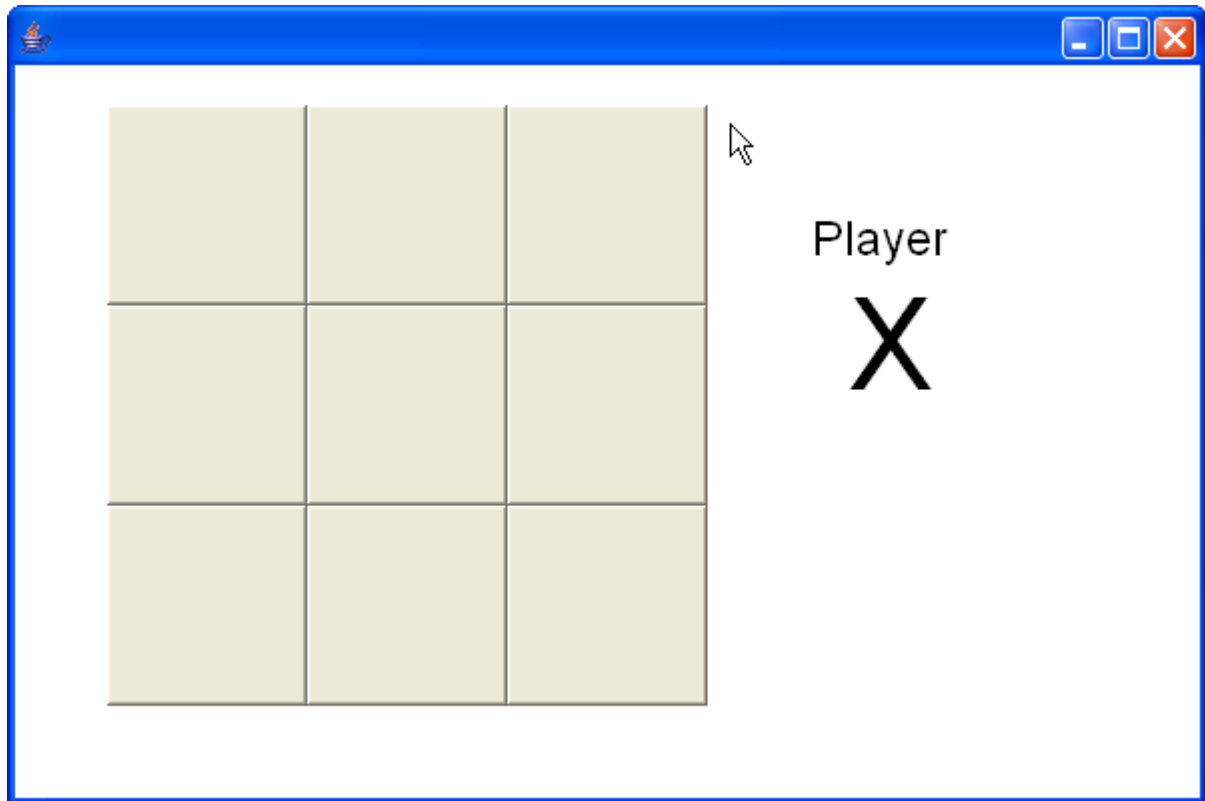
The initial design includes a data-file containing questions and the game module for playing the game.



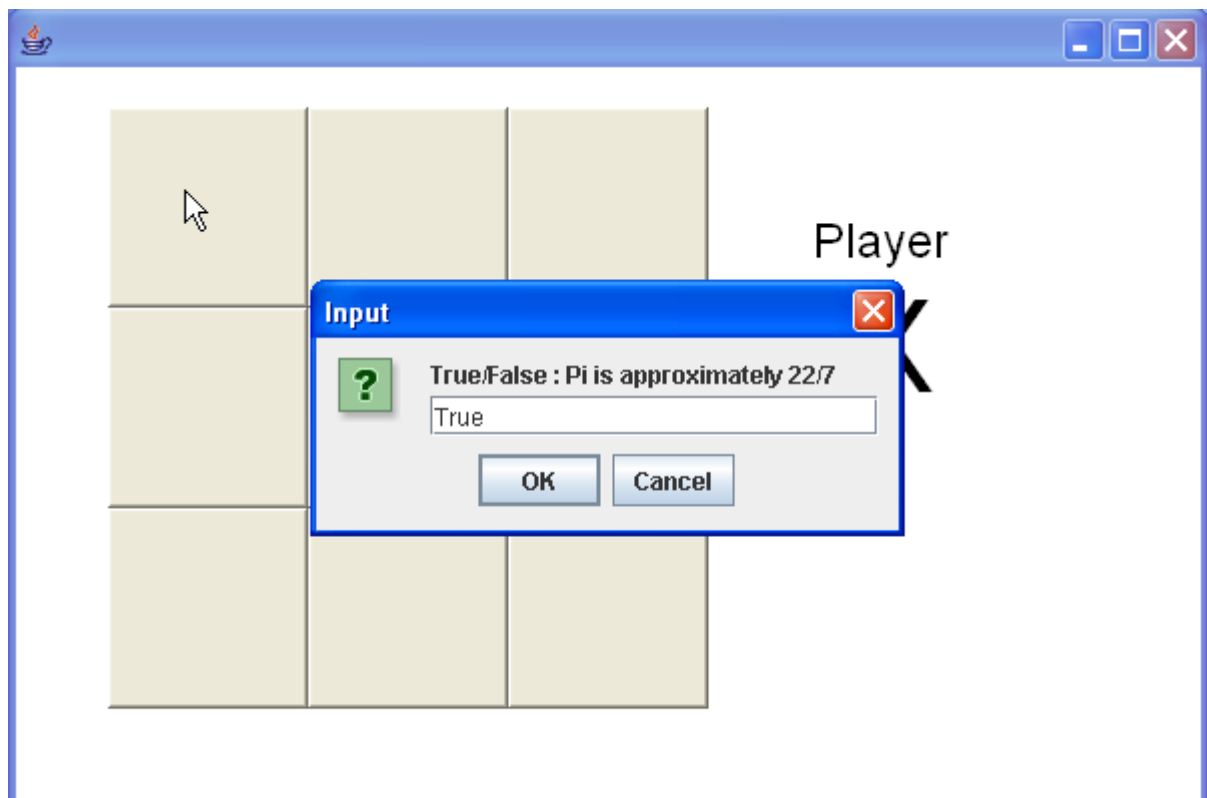
The finished program will need a large set of questions, but the prototype will have hard-coded questions for test purposes. So the Questions Module will not appear in the prototype – it is only simulated.

## Prototype – Sample Screens **\*\* the code listing is at the end of Stage A \*\***

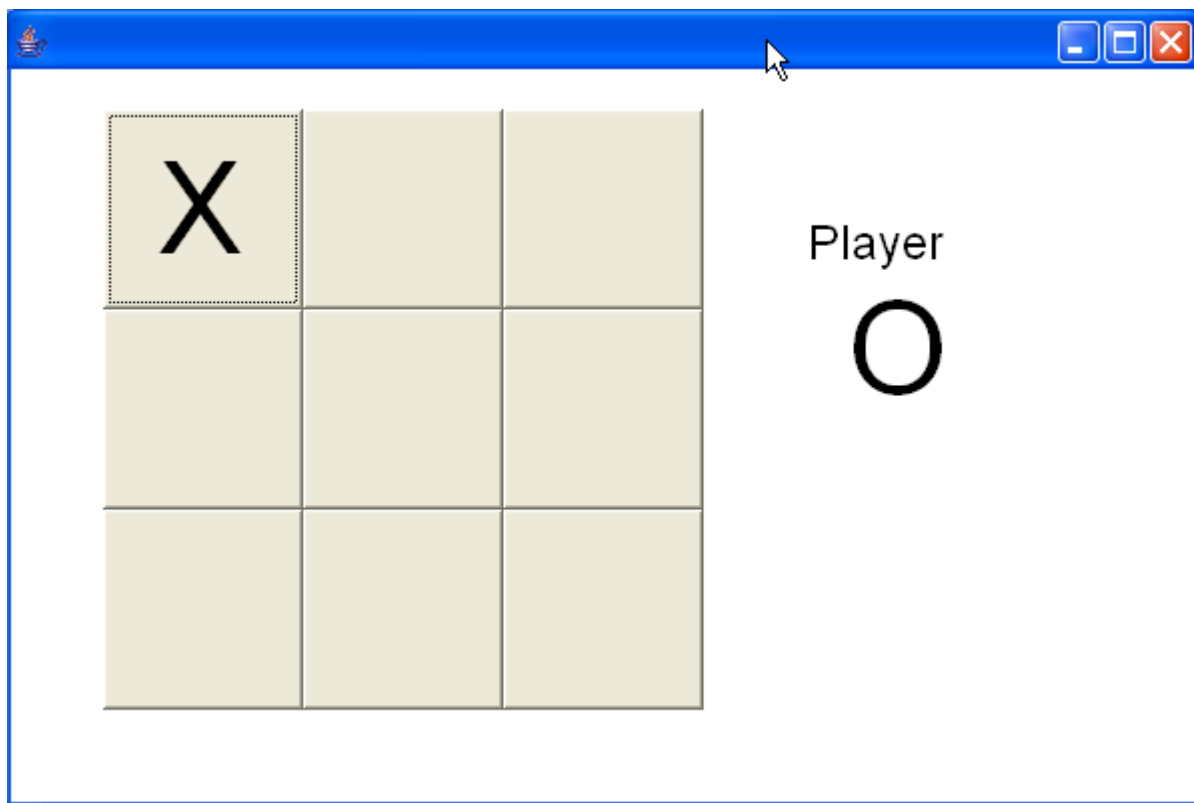
The quiz starts, X plays first.



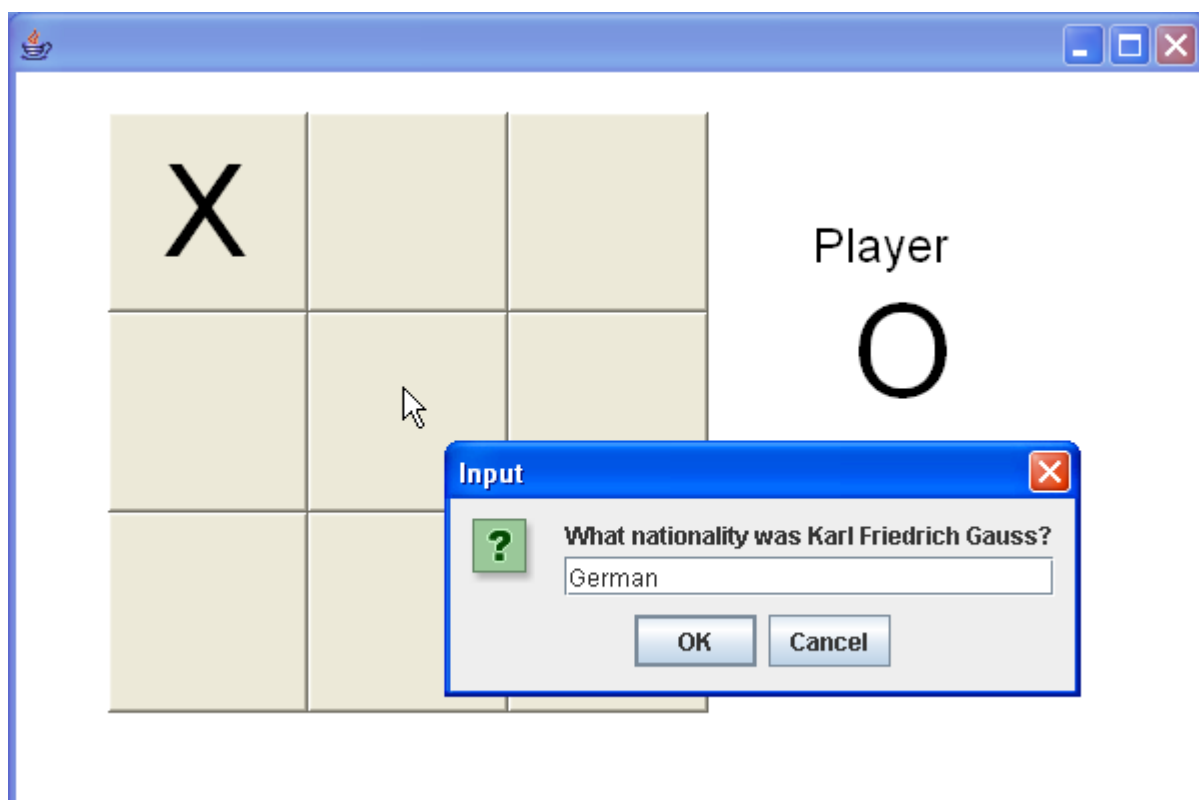
X clicks on the top-left corner and answers the question.



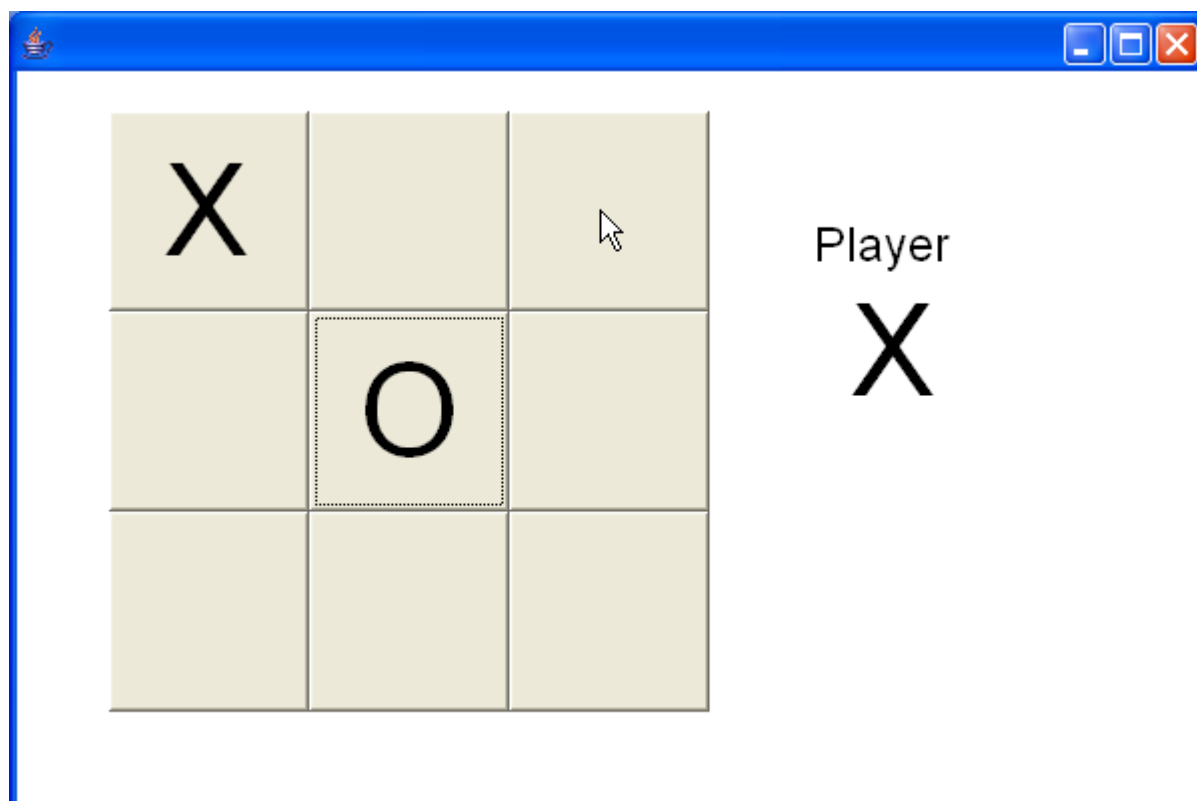
X had the correct answer, so gets the square. Now it is O's turn.



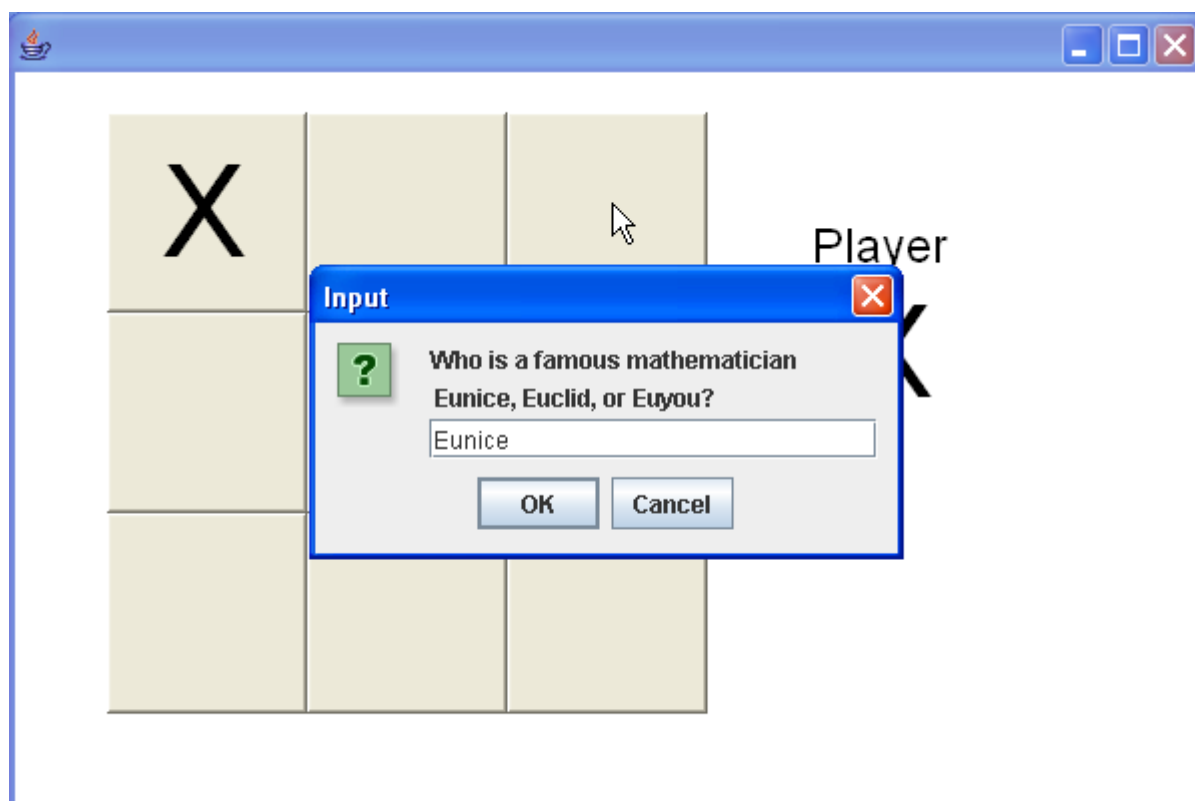
O chooses the middle square and answers the question.



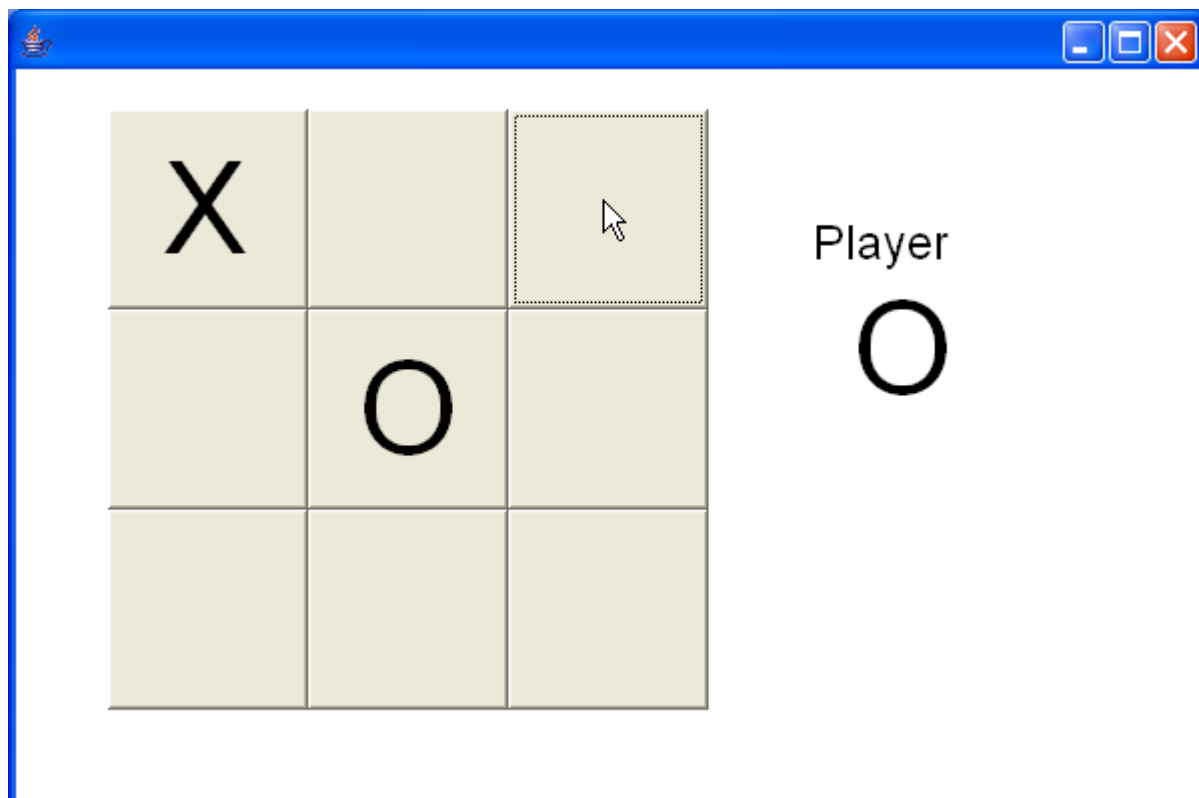
O answers correctly and gets the square.



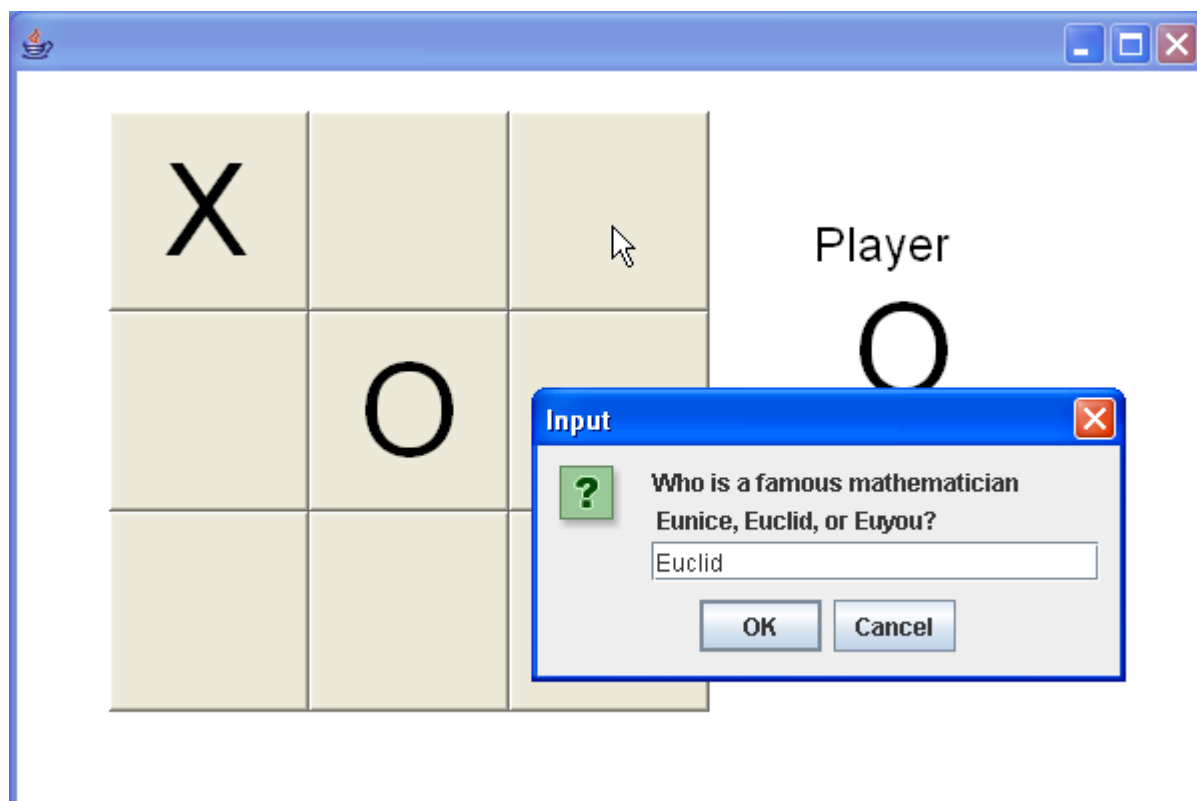
X tries for the top-right corner, but answers the question incorrectly.



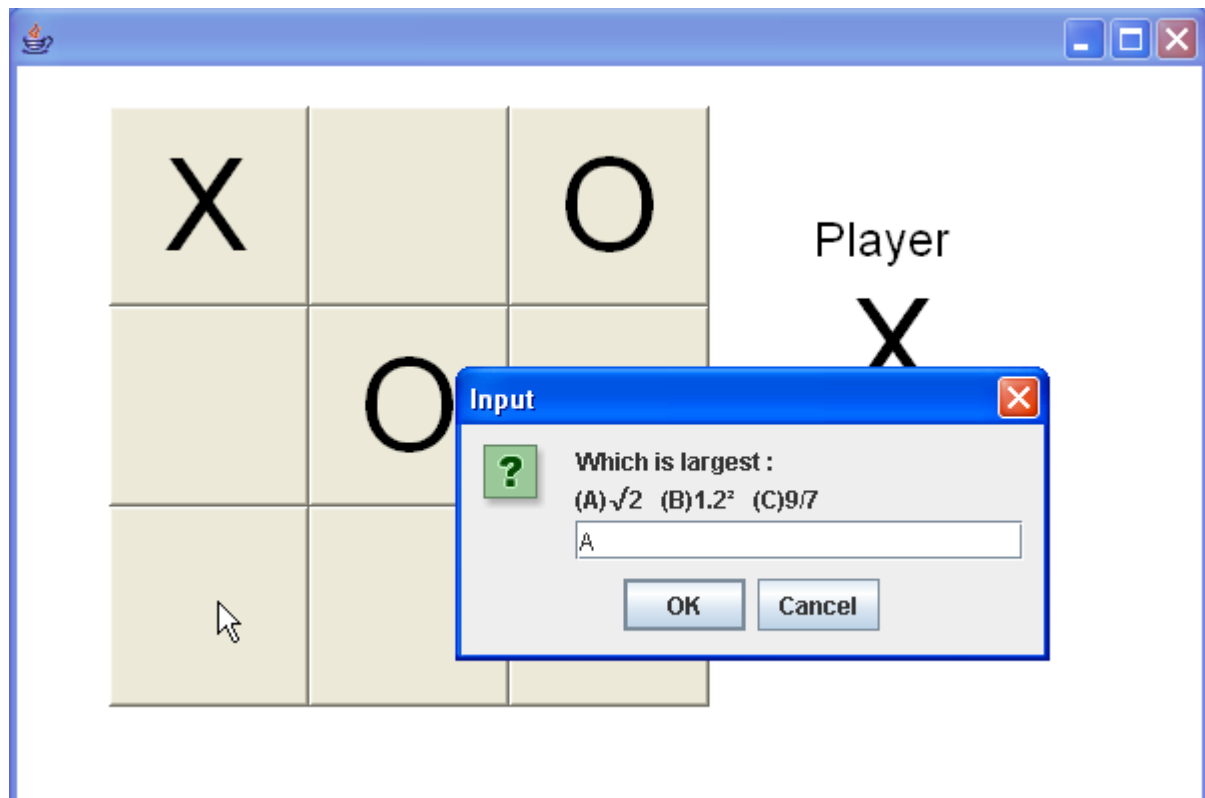
X answered incorrectly and thus didn't get the square. Now it's O's turn.



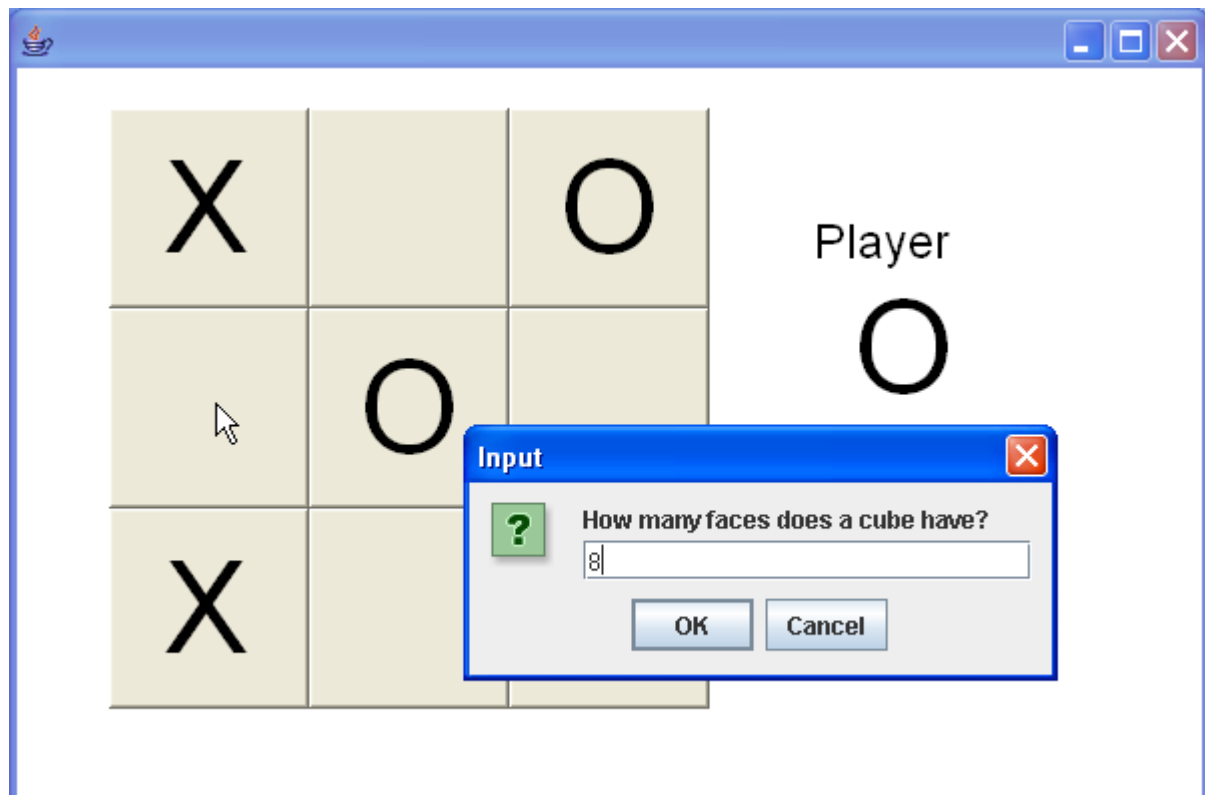
O tries the top-right square again, because the question was easy.



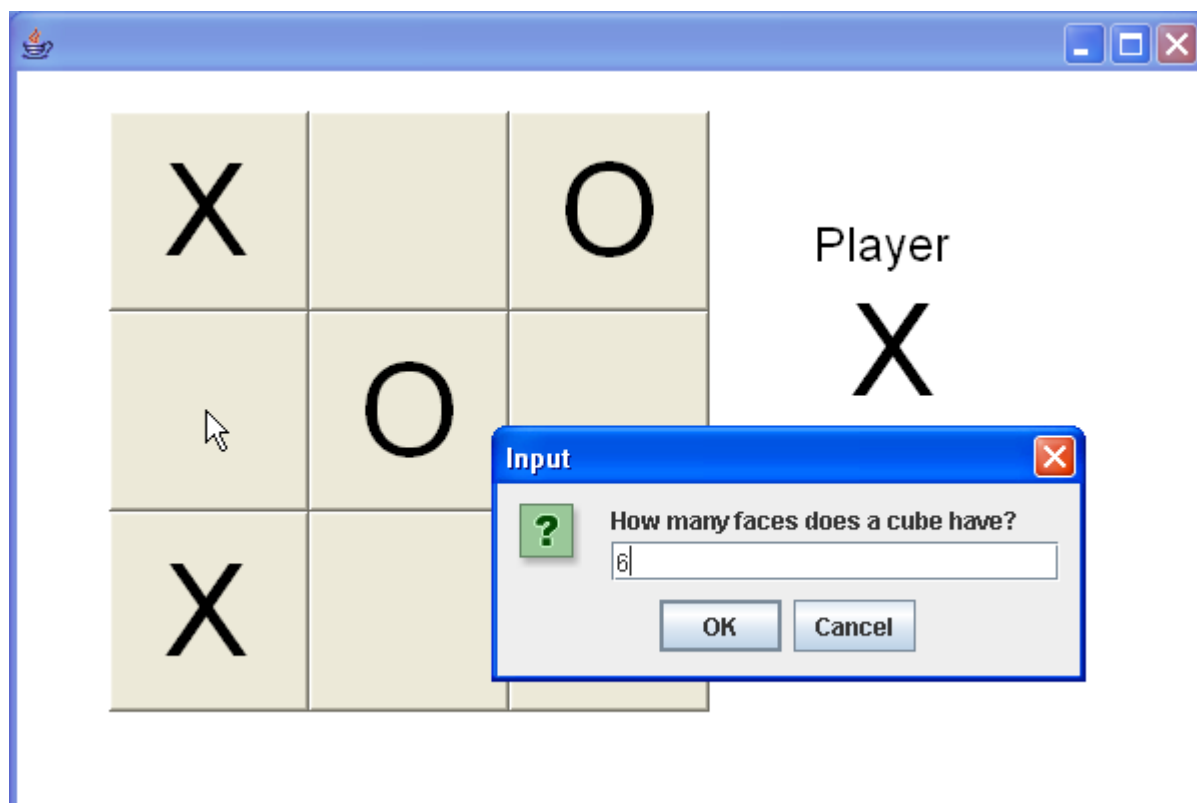
O answered correctly and got the square. Now it's X's turn. X goes for the block in the bottom left.



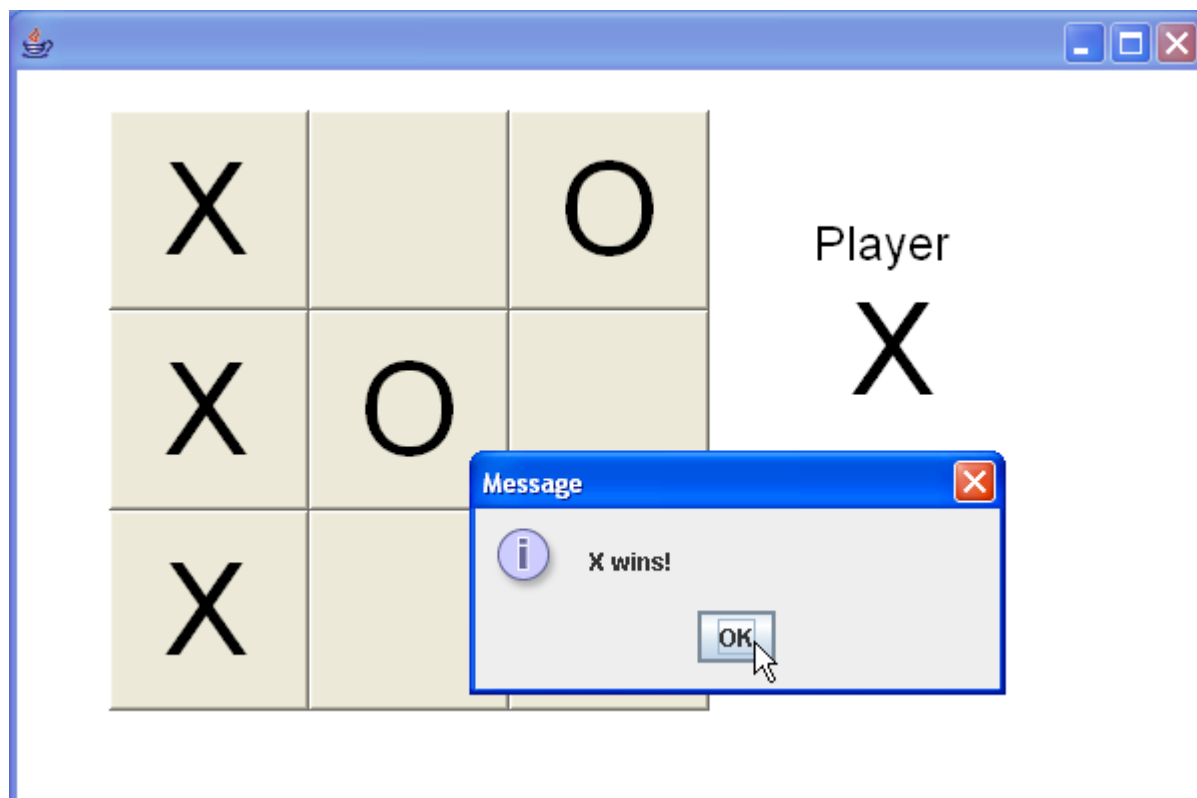
X had the right answer and gets the square. Now O goes for the block in the middle-left.



O answered incorrectly. Now X goes for the win.



X answered correctly and wins the game!



## User Feedback about the Prototype

We ran the prototype several times. Ms Fizz wanted to check whether the prototype worked for wins along the diagonals – it worked okay. Ms Fizz had many questions – below is an excerpt. The discussion was longer (several sessions actually), but these outline the issues affecting goals.

<i>User Questions, Ideas and Wishes</i>	<i>Answers</i>
Does it correctly recognize wins in all directions (including diagonals)?	Yes (we tested all the directions)
What happens if there is a tie – the board is full but no winner?	The prototype doesn't recognize a tie, but the final program will stop and say “It's a tie”. Is that okay?
If there is a tie, I guess the player with more squares should win.	Okay, we can do that.
This always has the same 9 questions. I think it should have different questions (selected randomly) each time you run it.	Okay, we can do that in the final program. Do you want the computer to invent the questions?
Can the computer invent the questions?	Not really, unless they are simple math calculations.
No, I want more flexibility in the questions. Some have words or formulas as answers, others are numbers.	Okay, then we need to make a module that lets you type in questions, then save them in a file, and the program loads the questions from the file.
I don't understand about saving in files. Is that hard?	No, we'll make a real simple interface and the saving and choosing random questions is all automatic.
But I want the questions scrambled up, so the players don't always know which question is hiding in each square.	If the questions are selected randomly, they'll also be arranged randomly in the squares.
I think in the old Hollywood Squares show, a wrong answer automatically gave the square to the other player. Is that possible?	Yes, it's possible – is that what you want?
I don't know – I'll think about it. Can I type real math formulas, with exponents and fractions?	No, I don't know how to program that. I guess you can type x-squared if you know the ASCII code for the squared sign, but no fractions.
Okay, I guess I can just type $3/4$ , like that. How about pictures – can I put in pictures?	I'll think about that. I don't know how to do it, but I'll ask my teacher if it's possible.
How many questions can I have altogether?	There is no limit. But it would be easier to write the program if we set some kind of limit – like 1000 questions maximum. Is that okay?
Is that a 1000 for EVERYTHING? I have lots of different classes, different each year.	We can make separate files for different subjects. What subjects do you need?
Let's see... simple algebra, advanced algebra, geometry, statistics, a few more. Maybe a 100 for each. How many can I have?	If you make a list of subjects and/or topics, we can make a set of questions for each. You can have any limit you like, but it's easier if the limit is fixed.
Okay, I'll think about it and make a list.	Can you make a list of topics and 10 sample questions for each? You can add more later.
Will this run on the school's web-site?	No, but it can run from a server on our LAN.

## Revised Criteria for Success after User Feedback

As a result of the user feedback, the original goals (Criteria for Success) were revised. Changes and additions are marked with \*\* asterisks.

### Criteria A2 – Criteria for Success (Final)

<b>Goal</b>	<b>Reason(s)</b>	<b>Limitation(s)</b>
Game plays Tic-Tac-Toe , using correct Tic-Tac-Toe rules	It's fun	3x3 board only
Each Tic-Tac-Toe square asks a question	That's the rules of the game	Questions will be selected from a list – not auto-generated
** Questions will be selected randomly and scrambled	Students should not know which questions are hiding in each box	
Questions should contain appropriate math content	Students should be learning and practicing their math	Some text-book problems are not suitable – e.g. complex formulas or graphs
Game should be quick, easy and satisfying, including a simple and clear user-interface	Motivate students to practice , more fun = more practice	Don't require complex input (fractions, complex formulas)
Teacher can create and save problems	Auto-generating problems is too difficult to program	No complex formulas Pictures?
Some special symbols can be used in the questions – squares, cubes, simple square-roots	Math without special symbols is difficult to read and understand	Many special symbols will not be implemented, especially fractions and complex surds like the discriminant in the quadratic formula
** Teacher module for typing and saving questions and answers	Teacher wants to create specific questions	Only a few special symbols Text only - no pictures
** Special symbols can be typed using keyboard shortcuts, e.g. ^2 for squared	Teacher does not want to learn ASCII codes	Shortcuts will be hard-coded and not changeable by the user
** Questions are saved into various files according to topic. Teacher should be able to add more topics (files) later	Teacher has various classes and topics	There will be a limit of 1000 questions per file
** Questions in data-files can be added, changed and deleted later	Teacher may need to make changes and corrections	This will not be drag-and-drop, but will function in text-mode after a simple search
** It should be easy to copy a problem, change a few numbers and then save as a new problem	Teacher wants to make several similar questions with slightly different numbers	It will be done in text-mode, not drag and drop

## Prototype Listing

```
import java.awt.*;

public class ProtoSquares extends EasyApp {
    public static void main(String[] args)
    { new ProtoSquares(); }

    Button[][] squares = new Button[3][3];
    Label lTurn = addLabel("Player",400,100,100,35,this);
    Label turn = addLabel("X",420,120,100,100,this);

    String[] questions =
    { "True/False : Pi is approximately 22/7",
      "What is the square root of 0.25?",
      "Who is a famous mathematician \n Eunice, Euclid, or Euyou?",
      "How many faces does a cube have?",
      "What nationality was Karl Friedrich Gauss?",
      "How many feet are there in one mile?",
      "Which is largest : \n(A)\u221a2      (B)1.22      (C) 9/7  ",
      "What is the root of  x2 - 8x + 16 ?",
      "What does 0! equal?"
    };

    String[] answers =
    { "True","0.5","Euclid","6","German","5280","A","4","1" };

    int player = 1;

    public ProtoSquares()
    { Font thefont = new Font("Arial",0,64);
      turn.setFont(thefont);
      lTurn.setFont(new Font("Arial",0,24));
      for (int row = 0; row < 3; row = row+1)
      {
          for (int col = 0; col < 3; col = col + 1)
          {
              int x = 50 + 100*col;
              int y = 50 + 100*row;
              squares[row][col] = addButton("",x,y,100,100,this);
              squares[row][col].setFont(thefont);
          }
      }
    }

    public void actions(Object source, String command)
    { int qnum = -1;
      int rnum = -1;
      int cnum = -1;

      for (int row = 0; row < 3; row = row + 1)
      { for (int col = 0; col < 3; col = col + 1)
        {
            if (source == squares[row][col])
            {qnum = row*3 + col;
              rnum = row;
              cnum = col;
            }
        }
      }
    }
}
```

```

    if (qnum >= 0)
    {
        if (squares[rnum][cnum].getLabel().equals(""))
        {
            String guess = input(questions[qnum]);
            if (guess.equalsIgnoreCase(answers[qnum]))
            {
                if (player == 1)
                {
                    squares[rnum][cnum].setLabel("X");
                }
                else
                {
                    squares[rnum][cnum].setLabel("O");
                }
                checkWinner();
                player = -1*player;
                if (player==1)
                {
                    turn.setText("X");
                }
                else
                {
                    turn.setText("O");
                }
            }
            else
            {
                output("Choose an empty square");
            }
        }
    }
}

public void checkWinner()
{
    for (int row = 0; row < 3; row = row + 1)
    {
        String a = squares[row][0].getLabel();
        String b = squares[row][1].getLabel();
        String c = squares[row][2].getLabel();
        if (!a.equals("") && a.equals(b) && b.equals(c))
        {
            output(a + " wins!");
            System.exit(0);
        }
    }
    for (int col = 0; col < 3; col = col + 1)
    {
        String a = squares[0][col].getLabel();
        String b = squares[1][col].getLabel();
        String c = squares[2][col].getLabel();
        if (!a.equals("") && a.equals(b) && b.equals(c))
        {
            output(a + " wins!");
            System.exit(0);
        }
    }
    String a = squares[0][0].getLabel();
    String b = squares[1][1].getLabel();
    String c = squares[2][2].getLabel();
    if (!a.equals("") && a.equals(b) && b.equals(c))
    {
        output(a + " wins!");
        System.exit(0);
    }
    String d = squares[0][2].getLabel();
    String e = squares[1][1].getLabel();
    String f = squares[2][0].getLabel();
    if (!d.equals("") && d.equals(e) && e.equals(f))
    {
        output(d + " wins!");
        System.exit(0);
    }
}
}
}

```

## Stage B1 - Data-Structures

The program will contain 3 sections (modules) :

**Students' Game Interface , Problem Storage , Teachers' Problem Interface**

### Problem Storage in Files

All the problems must be stored in **data-files** on a disk drive. The teacher wants separate problem lists for various topics. Each file must contain a record for each problem. Each record contains three fields : Question, Choices and Answer. If the question is longer, the Choices can contain part of the question instead of multiple answers, but then the user must type the exact answer.

A **multiple-choice** problem looks like this:

Question : What do you call a polygon with 8 sides?

Choices : (A) Eightogon (B) Stop Sign (C) Octagon (D) Octogon

Answer : C

A **type-the-exact-answer** question looks like this:

Question : What is the sum of the roots of this quadratic equation?

Choices :  $x^2 - 4x + 2 = 0$

Answer : 4

Text-files have the advantage that it is possible to make small corrections using a text-editor. However, they allow only sequential access. Since a small set of 9 questions must be selected at random from the file, a RandomAccessFile is more efficient as single problems can be accessed by their record number. Although RandomAccessFiles are a bit trickier than sequential files, the coding will probably be shorter in the long run with RandomAccessFiles.

The files will each contain 1000 records (fixed file size). Blank records will contain zero-length Strings. Each record will be 200 bytes, so 200 KiloBytes per file. T

Ms Fizz agreed to these shortcuts: ^2 for squared, ^3 for cubed, and ^r for a square-root sign  $\sqrt{\phantom{x}}$ .  
Like this: **Roots of  $x^2 - 6 = 0$  are \r6 , -\r6  $\implies$  Roots of  $x^2 - 6 = 0$  are  $\sqrt{6}, -\sqrt{6}$**

Each data-file will look something like this file for **Algebra2**:

Record #	Fields and Sizes	Sample Data
0	Question: 90 bytes Choices : 90 bytes Answer : 20 bytes	Which is a root of $2x^2 - 6 = 0$ ? (A) \r2 (B) \r3 (C) \r6 (D) \r12 B
1	Question: 90 bytes Choices : 90 bytes Answer : 20 bytes	Where is the vertex of this parabola? $y = x^2 + 4x + 4$ (-2,0)
....	....	....
999	Question: 90 bytes Choices : 90 bytes Answer : 20 bytes	"" "" ""

Ms Fizz has asked for the following topic files:

- Numbers (fractions, decimals, percents)
- Shapes (Circles, Rectangles, Triangles, etc)
- Statistics
- Equations (solving linear equations)
- Quadratics
- Graphs
- ... more to be added later ...

We agreed there should also be a feature to create a new topic. That will create an empty file with 1000 records. That way she can add more whenever she wants. Here are 2 sample files showing 5 questions each:

<b><i>Numbers</i></b>
Which decimal equals $\frac{3}{8}$ ? (A) 0.38 (B) 0.375 (C) 0.388... B
Which fraction is largest? (A) $\frac{3}{4}$ (B) $\frac{4}{5}$ (C) $\frac{7}{8}$ C
What is $\frac{2}{3} + \frac{1}{12}$ ? (A) $\frac{3}{4}$ (B) $\frac{3}{15}$ (C) $\frac{9}{12}$ A
Which number is the largest? (A) Billion (B) Million (C) Googol C
Calculate 30% of 30. 9
....
....
....

<b><i>Quadratics</i></b>
Which is a root of $2x^2 - 6 = 0$ ? (A) $\sqrt{2}$ (B) $\sqrt{3}$ (C) $\sqrt{6}$ (D) $\sqrt{12}$ B
Where is the vertex of this parabola? $y = x^2 + 4x + 4$ (-2,0)
Fill in the blank to “complete the square” $x^2 - 6x + \underline{\hspace{1cm}}$ 9
Solve : $2x^2 - 8x + 6 = 0$ (A) 1 , 3 (B) -1 , -3 (C) 2 , 6 A
True or false: The Vertex of $x^2 - 15$ is located directly on the x-axis. True
....
....
....

## Teacher Interface - Problems Stored in a Problem Class

The teacher must add new problems to the problem files. The program needs to control the input to prevent the teacher typing text that's too long for the 90 byte fields in the file. It must also convert shortcut codes to proper ASCII characters. The simplest way to do this is to create a data-storage Class called Problem. This class can store the 3 fields for a single problem (Question, Choices, Answer) and also ensure that these fields contain valid data by implementing **accessor** methods (**get** and **set** methods). The class will look something like this:

```
public class Problem
{
    private String question = "";
    private String choices  = "";
    private String answer   = "";

    public boolean setQuestion(String q)
    {
        //  replace shortcuts \r , ^2 , ^3 with ASCII characters
        //  check that q.length is under 88 bytes
        //  if not, return false as a rejection message
        question = q;
        return q;
    }

    public boolean setChoices(String c)
    {
        // similar to setQuestion
    }

    public boolean answer(String a)
    {
        // similar to setQuestion, but length must be under 18
    }

    public String getProblem() { return question; }
    public String getChoices() { return choices; }
    public String getAnswer()  { return answer; }
}
```

## Game Interface - Arrays of Problems and Buttons

### Set of 9 Problems in 1-D Array

The Game Interface module can use the same **Problem** class for storing problems. Each game must select 9 random problems from the Files discussed above. So it will use those same files. After selecting 9 random problems, the problems can be stored in an array - an array of Problem objects:

```
Problem[] problems = new Problem[9] ;    // 1-dimensional array
```

The data will be the same as the data stored in the files (see above for sample data).

This array must be scrambled up randomly before starting the game - this can be done by swapping two random locations and repeating that hundreds of times. It's like shuffling cards.

### GUI Buttons in a 1-D Array

Although the Tic-Tac-Toe board is in the shape of a 2-dimensional array, the coding for this is unnecessarily long. It's simpler to make a 1-dimensional array of Buttons.

```
Button[] squares = new Button[9] ;    // 1-dimensional array
```

<b>square[0]</b> at 50,50	<b>square[1]</b> at 150,50	<b>square[2]</b> at 250,50
» problem[0]	» problem[1]	» problem[2]
<b>square[3]</b> at 50,150	<b>square[4]</b> at 150,150	<b>square[5]</b> at 250,150
» problem[3]	» problem[4]	» problem[5]
<b>square[6]</b> at 50,250	<b>square[7]</b> at 150,250	<b>square[8]</b> at 250,250
» problem[6]	» problem[7]	» problem[9]

This approach is different than the prototype. It requires a bit of care in calculating the coordinates for the positions of the buttons - coordinates are shown in the diagram above.

The checkWinner( ) method is also a bit different - it's actually shorter and easier to code, but requires a bit more thought than a 2-D array. Checking the columns for a winner will look something like this (it's actually less code than the prototype):

```
for (int col = 0; row < 3; row = row + 1)
{
    String a = squares[col].getLabel();
    String b = squares[col+3].getLabel();
    String c = squares[col+6].getLabel();

    if (!a.equals("") && a.equals(b) && b.equals(c))
    {
        output("Player " + a + " wins" );
    }
}
```

This also turns the **problems[ ]** and **squares[ ]** arrays into **parallel arrays**, making lots of the coding simpler. For example, if **squares[3]** is clicked, then **problems[3]** is displayed.

